Mining useful Macro-actions in Planning

Sandra Castellanos-Paez, Damien Pellier, Humbert Fiorino, Sylvie Pesty

Mining useful Macro-actions in Planning

Sandra Castellanos-Paez, Damien Pellier, Humbert Fiorino and Sylvie Pesty
Univ. Grenoble Alpes, LIG, F-38000
Grenoble, France
Email: firstname.lastname@imag.fr

***Abstract*—Planning has achieved significant progress in recent years. Among the various approaches to scale up plan synthesis, the use of macro-actions has been widely explored. As a first stage towards the development of a solution to learn on-line macro-actions, we propose an algorithm to identify useful macro-actions based on data mining techniques. The integration in the planning search of these learned macro-actions shows significant improvements over six classical planning benchmarks.**

I. Introduction

Automated planning is an area of Artificial Intelligence that comes up with the challenge of devising systems that can autonomously find a plan to reach a set of goals. In classical planning, a problem is composed of an initial state, a goal specification and a set of actions. From the initial state if the preconditions of an action are satisfied, the action is applicable to the current state. Thereby, the action effects can be applied to generate a new state. This is done for each new state until the goal is reached. The solution of a planning problem is a plan. A plan defines a sequence of actions from the initial state to the goal state for a given problem over a domain.

Planning is NP-hard. Thus, the focus remains on developing powerful planning techniques capable of effectively explore the search space that grows exponentially. A way to increase planner performance consists in exploiting knowledge about the structure of the planning tasks [1]. This could be accomplished by using *macro-actions*, i.e., a sequence of actions that occurs frequently in solution plans. Macro-actions have been widely studied among the different approaches to speed-up planning processes. Learning macro-actions from previously acquired knowledge (plans) allows to go deep quickly into the search space by triggering them during the search.

The use of macro-actions is not arbitrary. If a sequence of actions has a higher frequency on different plans, reasoning lead us to consider it as a candidate of useful sequences for a given domain. Let's have an example, consider the *blocksworld* planning domain in Figure 1. The goal is to stack a set of blocks. This domain has five operators: pick-up, picks a block x from the table; put-down, puts a block x on the table; stack, puts a block x on a block y; unstack, removes a block x from a block y. It is logical to suggest that once we pick a block from the table the next most probably action will be stack it on another block or put it down. If we learn one of these sequences as a macro-action e.g. pick-up_stack, we can apply it directly avoiding the analysis of what action comes after the pick-up action.

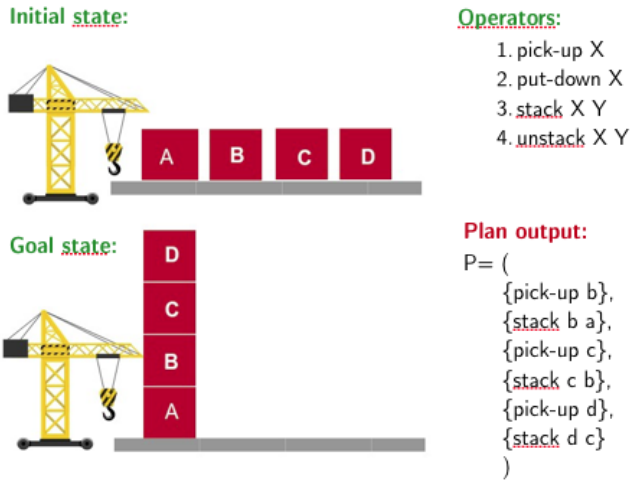

Fig. 1. Blocksworld domain.

Figure 2 shows our framework to learn useful sequences as macro-actions and use them to speed-up the search. We solve some problems and we keep the output plans in order to build a sequence database (Figure 3). Then, we look for useful sequences by using a sequential pattern mining algorithm. This algorithm finds the complete set of patterns satisfying a minimum frequency threshold. Every frequent sequence of that set represents a macro-action. Finally, we feed the planner with the macros to save time searching.

Although much work has been done to date [2], [3], [4], [5], [6], more studies need to be conducted to ensure successful results in the performance of planning engines in exploiting macros regardless of the planning domain. The purpose of this research was to further investigate a strategy for detecting useful macros based in pattern mining algorithms. We propose a novel planner independent macro learning method. First, to detect automatically useful macro-actions trough sequential pattern mining algorithms and second, to implement these macro-actions in any state-space search algorithm to speed-up the search.

The paper is organized as follows. We will first introduce a literature review in macro-actions past works. Then we will present the planning concepts used in this work followed by the motivation for using pattern mining algorithms. After that, we will get into the core of the proposed method for generating and using macro-actions. Finally, we will discuss the results and the possible future work.

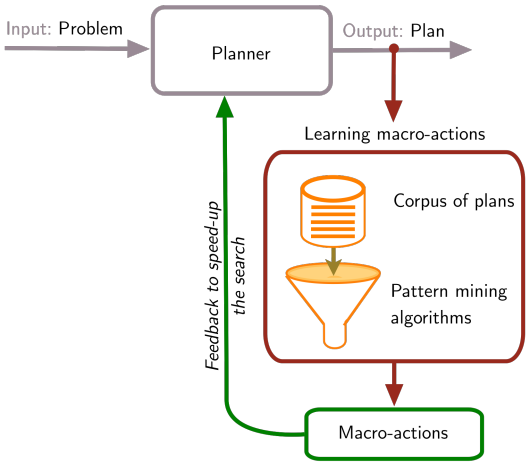

Fig. 2. Mining Framework.

ID	Plans
Seq1	{pick-up b}, {stack b a}, {pick-up c}, {stack c b}, {pick-up d}, {stack d c}
Seq2	{unstack b c}, {put-down b}, {unstack c a}, {put-down c}, {unstack a d}, {stack a b}, {pick-up c}, {stack c a}, {pick-up d}, {stack d c}
Seq3	{unstack c b}, {stack c d}, {pick-up b}, {stack b c}, {pick-up a}, {stack a b}
Seq4	{unstack b a}, {stack b c}, {unstack a d}, {stack a e}, {unstack b c}, {stack b a}, {pick-up c}, {stack c b}, {pick-up d}, {stack d c}

Fig. 3. Corpus of plans.

II. Related Work

We group macro-actions related work into two main categories: off-line and on-line techniques. An off-line approach offers as an advantage an ease view over the macro-actions use, but also over the impact in the search time. Into this category, we found the adaptive planning system Macro-FF [2]. It extracts macro-actions from solutions of training problems, and by identifying statically connected abstract components. Only the macro-actions showing effective performances in solving training problems are kept for future searches. Newton et al. [3] proposed another offline method which uses a genetic algorithm as a learning technique and plans as the macro generation source. The algorithm generates the macros from plans of simple problems to seed the population and evaluates them through a ranking method based on the weighted average of time differences in solving more difficult problems with the original domain augmented with macros. In a more recent work, Dulac et al. [5] introduced a domain independent approach for learning macros from before computed solutions. It extracts statistical information from successful plans based on a n-gram analysis. Then it builds a macro library based on earlier information, a generalisation and a specialisation process. Finally, it adds selected macros into the planning domain after a filtering phase based on statistical information and heuristics. Later, Chrpa et al. proposed a technique [6] to maximise the utility of macros. It first learns the causal relations between planning operators and initial or goal predicates (also known as outer entanglements) by using an approximation algorithm in several training plans. Then, exploiting this knowledge it extracts macros and uses them to reformulate the original domain model.

By contrast, an on-line approach remove the need of extra training problems and off-line filtering. Coles and Smith described Marvin planner in [7]. It identifies regions in the search space where the heuristic values of all successors is greater than or equal to the best seen so far. Then, it learns the escaping macro-actions to use them in similar regions during the search. This work was improved in [8] by keeping libraries of macro-actions for use on future problems. Chrpa et al.[9] extended their early technique by generating useful macros from outer entanglements in the search without an offline learning phase.

Other approaches presented an algorithm [4] to decompose a domain in several subproblems to then solve them. After, the algorithm stores each generated partial plan in memory as a macro which allows to retrieve and use it as part of the solution for a different problem that contains it. Masataro and Fukunaga introduced a similar work in [10] which automatically identifies subproblems, generate macros from subplans and integrate the subplans by solving the augmented problem.

In spite of macro-actions can be intuitively built from sequence of actions occurring many times in solution plans, there is no previous work based on mining frequent sequences through data mining techniques. Also, no previous research on using macro-actions in automated planning has fully implemented a planner independent approach.

III. Pattern Mining for Macro Learning

A. Planning system definition

We address sequential planning in the STRIPS framework [11]. An *action* a is a tuple $a = (pre(a), add(a), del(a))$ where $pre(a)$ are the action's *preconditions*, $add(a)$ and $del(a)$ are respectively its positive and negative *effects*.

A *state* s is a set of logical propositions. A state s' is reached from s by applying an action a according to the transition function

$$\gamma(s, a) = (s - del(a)) \cup add(a).$$

The application of a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ to a state s is recursively defined as $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_n \rangle)$.

A *planning domain* is composed of a set of actions and a set of predicates i.e. properties of objects. A *planning problem* is defined by a planning domain, an initial state and a set of goal states. Thus, a *plan* is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ such that the goal $g \subseteq \gamma(s, \pi)$ and g is reachable if such a plan exists.

Planning problems have been shown PSPACE complete, it means the size of the search space is huge. Thus planning systems must reduce the size of the search space they traverse.

Classical approaches get as output a sequence of actions (plan) for every input problem from an input domain. However, for every new problem they execute the same process without keeping memory of past events. Real world problems give

us an idea of encapsulating routines in few steps that have subroutines inside. For example, in everyday life if we are in a room and we need go to the kitchen a possible solution is $(open-door-room, go-out, walk, open-door-kitchen)$. In the same context if we need go to the bathroom a possible solution is $(open - door - room, go - out, walk, open - door - bathroom)$. By examining common sequences of actions when solving problems in a given domain, we can generalize a routine. In our example, we could rewrite the solutions as $go - from - roomA - to - roomB$.

B. Interest towards pattern mining use

The discovery of recurring relationships among huge amount of data can help in the prediction of the next element in a sequence. *Frequent patterns* are patterns that appear frequently in a data set. A *sequence database* is a set of sequences where each sequence is a list of itemsets. A *sequential pattern* is a sequence $s_a = x_1, x_2, \ldots, x_k$ (where $x_1, x_2, \ldots, x_k$ are itemsets) is said to occur in another sequence $s_b = y_1, y_2, \ldots, y_m$ (where $y_1, y_2, \ldots, y_k$ are itemsets) if and only if there exists integers $1 \leq i1 < i_2 < \ldots < i_k \leq m$ such that $x_1 \subseteq y_{i1}, x_2 \subseteq y_{i2}, \ldots, x_k \subseteq y_{ik}$. The *support* of a sequential pattern is the number of sequences where the pattern occurs divided by the total number of sequences in the database.

A sequential pattern may have several forms: a *frequent sequential pattern* is a sequential pattern having a support no less than the parameter provided by the user; a *closed sequential pattern* is a sequential pattern that is not included in another pattern having the same support; a *maximal sequential pattern* is a sequential pattern that is not strictly included in another closed sequential pattern. We are interested in the latter because the set is much smaller than the others.

Table I presents the studied algorithms of sequential pattern mining for this work. Besides sequential rule mining algorithms were also analyzed but discarded because we are strictly focused in ordered sequences.

Algorithm	Form	Gap Config.	Max Length
CM-SPADE	all	no	no
CM-SPAM	all	yes	yes
BIDE+	closed	no	no
VMSP	maximal	yes	yes
MaxSP	maximal	no	no

TABLE I
SEQUENTIAL PATTERN MINING ALGORITHMS

Detecting sequences of actions in a set of plans can be compared with mining sequential patterns in a data set. However, our interest remains in finding optimal sets of ordered sequences of actions where each consecutive action of a pattern must appear consecutively in a sequence i.e. no gap is allowed. Only the algorithms allowing the gap configuration interest us. Otherwise if the gap configuration is not allowed then the algorithms could find frequent sequences where actions are not consecutive. With regard to this, we focused in VMSP[1] algorithm [12].

VMSP mines a compact representation of the set of sequential patterns making it easier for users to analyze results. It uses a depth-first exploration of the search space using a vertical representation. First, it implements a search procedure to construct the set of frequent sequences F_1 given a *minsup* threshold. Then, it filters the non maximal patterns from F_1 by checking from each sequence s_a if there exists a pattern s_b such that $s_a \sqsubseteq s_b$. Finally, it does a candidate pruning by exploiting item co-occurrence information.

C. Implementation

The main idea of our approach is to build macro-actions from sequential patterns of actions in a set of plans and to use them during the planning search to improve the performance of the planning system.

A *sequential pattern of actions* is a frequent action subsequence (α) existing in a single plan or a set of plans (θ). The support of a sequence $supp^{\theta}(\alpha)$ is the number of plans in θ that contains α. Given a support threshold σ, a sequence α is a *frequent sequence* on θ if $supp^{\theta}(\alpha) \geq \sigma$. Mining of maximal patterns consists of finding the set of *maximal sequences* i.e. for a sequence α there is no other sequence β such that $\alpha \sqsubseteq \beta$.

Macros should appear many times in solutions plans and encapsulate knowledge that is reusable across the problems in a domain. The first condition can be accomplished by studying the support parameter. If a sequence of actions occurs many times in plans, it might be a good sequence to examine. On the other hand, the second condition can take advantage of maximal sequences. A large number of sequences of actions makes it difficult to analyze results, then mining maximal sequences leads to a more compact set but also better efficiency.

Our method to learn and to use macro actions includes a *set-up algorithm* and a *enhanced search algorithm*. Algorithm 1 takes as input a learning set of non-empty solution plans. In line 3, it obtains a set of maximal sequential patterns by using the VMSP[12] algorithm. Afterwards, from a encoded problem P the function encodeMacros evaluates each sequence from R. It checks if each one of the actions from the evaluated sequence belongs to the encoded operators of the problem. When the condition is met the whole sequence is encoded and added in the problem macro-action list (line 7).

Algorithm 1 Set-up algorithm

```
1: D ← Set of non-empty solution plans
2: A ← Algorithm for maximal sequence mining
3: R ← SPMF(D, A, supp)
4: function ENCODEMACROS(encodedProblem P)
5:     for each action c in GetCandidate(R) do
6:         if c ∈ GetOperators (P) then
7:             T[indexOf(R)] ← Add (O_i)
8:     return T
```

[1]Vertical mining of Maximal Sequential Patterns

The Algorithm 2, takes as input the encoded macro-actions list obtained in Algorithm 1. This algorithm can be implemented to any search algorithm in trees to speed-up the search. The purpose of this work is to validate the added value of this generic method. For this work, we chose a classical implementation of the A* algorithm. Line 2 represents a priority queue while lines 3 and 4 represent respectively the explored and pending nodes. In line 6, a node x is selected from the pending nodes list taking into account its heuristic value h. A plan is reached when x satisfy the goal. If not the applicability of each macro-action $m_i = \langle a_1 a_2 ... a_n \rangle$ over x is evaluated in line 11. A macro-action is applicable in x when the preconditions of x satisfies the preconditions of $m_i[0]$ allowing to get the successor x' and for each obtained successor the next actions are applicable (from $n > 0, m_i[n-1]$ is applicable to x^{n-1}). The created successors update the list of pending nodes and the list of explored nodes (lines 13 and 16). After the algorithm tries to apply the problem operators. Finally, the node x is added to the explored nodes and another node is selected from the pending nodes list.

Algorithm 2 Enhanced search algorithm

Input macro-actions list $T[m_i < a_1, ..., a_n >]$

```
1:  P ← {}
2:  open ← root
3:  closeset ← {}
4:  openset ← init
5:  while open ≠ null do
6:     current ← poll(open)
7:     if current satisfy G then
8:        extract (P)
9:     else
10:       for each macro m_i[a_1, a_2, ..., a_n] in T do
11:          if m_i isApplicableTo current then
12:             for each s_i in generateStates(current, m_i) do
13:                if s_i ∈ openSet and cost(s) < cost (s_i) then
14:                   update (s, s_i)
15:                else
16:                   if s_i ∈ closeSet and cost (s)< cost (s_i) then
17:                      remove (closeSet, s_i)
18:                   add (openSet, open, s_i)
19:       applyOperators (current)
```

Output P

IV. Results and Discussion

A. Experimental setup

The experiments were based on barman, blocksworld, depots, ferry, grid and sokoban benchmarks. They were carried out on an Intel Xeon E5-2630 2.30GHz. The allocated CPU time was set to 300 seconds with a maximum of 8GB of memory. For each benchmark, a learning set of plans of 1000 problems and a test set of 300 problems were randomly generated with the generators[2] used for the International Planning Competition (IPC). We went trough the SPMF [13] data mining library, which implements the VMSP algorithm, to get the set of maximal sequences. We did not allow gaps and we varied the degree of support in steps of 1 from 1% until no sequences were founded. We used the PDDL4J [3] library to encode the sequences into a forward chaining planner based on A* algorithm and on FF heuristic [14].

B. Evaluation criteria

The evaluation was based on the classical metrics of quality and time used in IPC. Time score is computed as the quotient $T*/T$ where $T*$ is the minimum time required by the planner to solve the problem, and T is the time spent by the evaluated implementation to solve the same planning task. Quality score is computed as $Q*/Q$ where $Q*$ is the cost of the best known plan for a particular problem and Q is the cost of the plan produced by the evaluated implementation. If the planner found no solution the quality is set to zero.

C. Results

Figures 4, 5, 6, 7, 8 and 9 report the gain of our method compared with the original algorithm among different support values. We can observe that large gains are obtained using lower values of the support parameter. TableII presents the results as relative gains with respect to the original algorithm given a support of 1%.

Domain	Time	Quality
Barman	372%	78%
Blocksworld	65%	-4%
Depots	34%	4%
Ferry	221%	-6%
Grid	595%	0%
Sokoban	142%	-12%

TABLE II
IMPROVEMENT WITH 1% SUPPORT

D. Discussion

Macro-actions quality score was improved in some domains owing the fact that the original algorithm could not found a plan for some problems (barman, blocksworld and depots). However, a domain where all the problems were solved by the original algorithm shows the worst quality (sokoban). It is due to the utility problem, i.e., the increasing of the branching factor due to the adding of macro-actions.

In our results, time improvement is obtained with a support of 1% for all the domains. The results suggest that: (1) it can be possible to identify potential macro-actions over a domain by fixing the degree of support between 1% and 30%; (2) search performance can be improved, thus validating the relevance of macro-actions learning in the planning search. (3) the use of pattern mining algorithms is a good strategy to efficiently identify macro-actions.

[2]https://bitbucket.org/planning-tools/pddl-generators
[3]https://github.com/pellierd/pddl4j

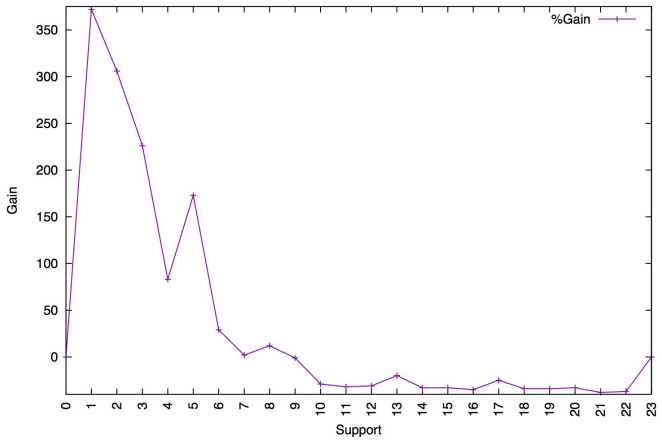

Fig. 4. Barman domain.

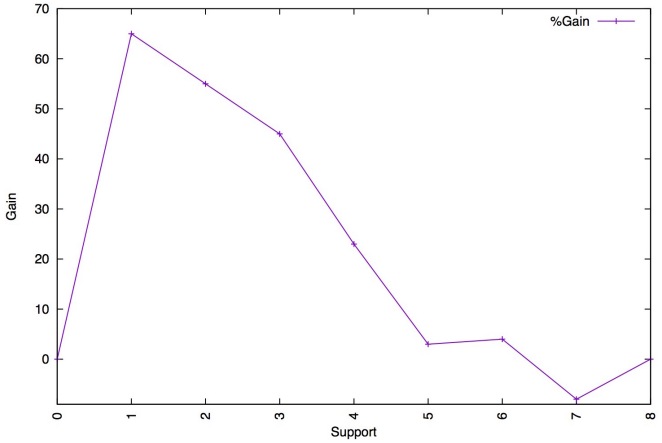

Fig. 5. Blocksworld domain.

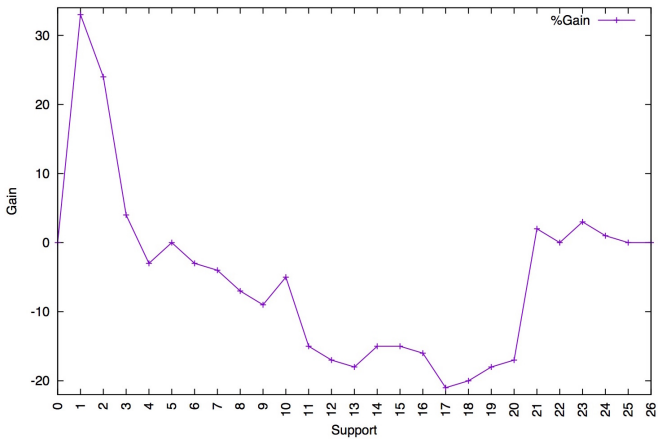

Fig. 6. Depots domain.

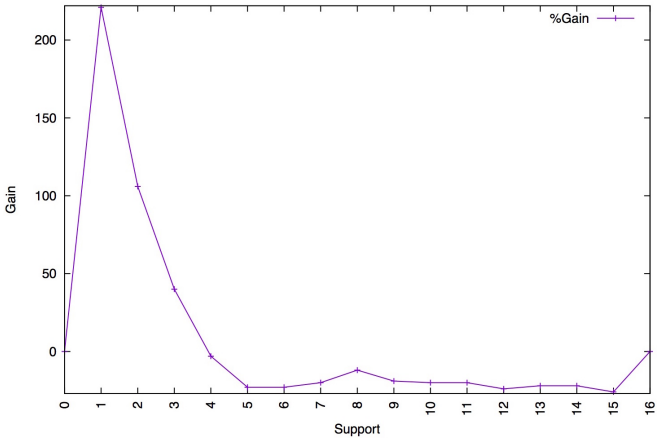

Fig. 7. Ferry domain.

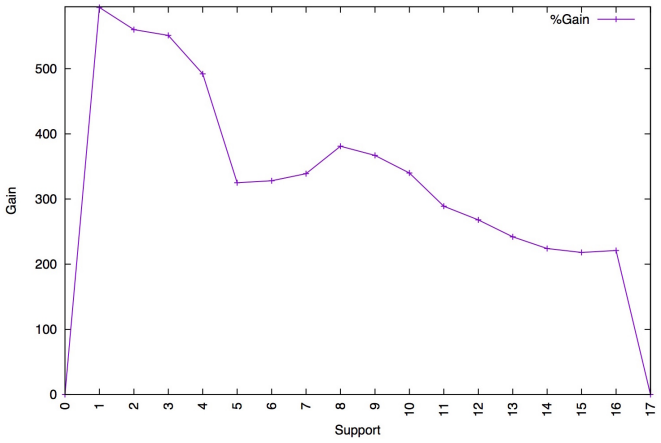

Fig. 8. Grid domain.

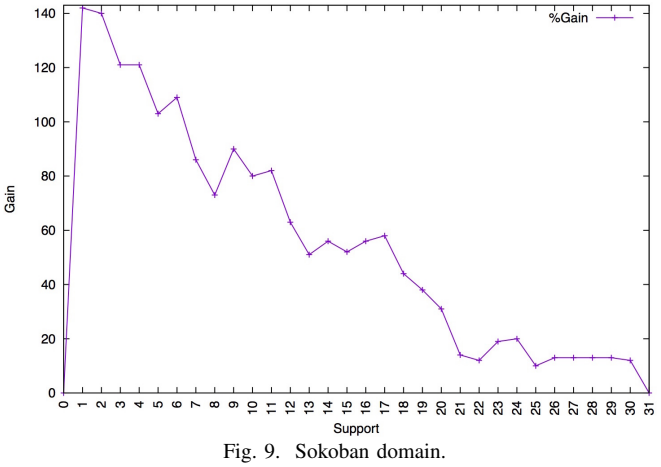

Fig. 9. Sokoban domain.

Further work includes:

- The formalization of useful macro-actions as macro-operators based on the generalization of frequent sequences.
- Implementing in our approach a way to deal with the utility problem. One promising direction is to apply outer entanglements [9] on macro-actions to reduce the number of instances of macros exploited.
- The development of a solution to learn on-line macro-actions.

V. Conclusion

We proposed a novel planner independent macro learning method. It starts by automatically detecting useful macro-actions trough sequential pattern mining algorithms. We give special attention to the VMSP algorithm which finds maximal sequences as optimal sets of ordered sequences of actions. It then implements those macro-actions in a forward chaining planner based on A* algorithm and on FF heuristic.

We have demonstrated that our method can be used to speed-up the search with a solid, and sometimes dramatic, gain

compared to not using macro-actions. This stands for the six studied domains, specially Grid, Barman, Ferry and Sokoban.

REFERENCES

[1] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis," *J. Artif. Int. Res.*, vol. 20, no. 1, pp. 1–59, Dec. 2003.

[2] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer, "Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators," *Journal of Artificial Intelligence Research*, vol. 24, pp. 581–621, 2005.

[3] M. A. H. Newton and J. Levine, "Implicit Learning of Macro-Actions for Planning," in *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, August 2010.

[4] A. Jonsson, "The Role of Macros in Tractable Planning," *J. Artif. Int. Res.*, vol. 36, no. 1, pp. 471–511, Sep. 2009.

[5] A. Dulac, D. Pellier, H. Fiorino, and D. Janiszek, "Learning Useful Macro-actions for Planning with N-Grams," in *IEEE 25th International Conference on Tools with Artificial Intelligence*, Nov 2013, pp. 803–810.

[6] L. Chrpa, M. Vallati, and T. L. McCluskey, "MUM: A technique for maximising the utility of macro-operators by constrained generation and use," in *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.

[7] A. Coles and A. Smith, "Marvin: A heuristic search planner with online macro-action learning," *Journal of Artificial Intelligence Research*, vol. 28, pp. 119–156, 2007.

[8] A. I. Coles, M. Fox, and A. J. Smith, "Online identification of useful macro-actions for planning," in *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 07)*, September 2007.

[9] L. Chrpa, M. Vallati, and T. McCluskey, "On the online generation of effective macro-operators," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, Q. Yang and M. Wooldridge, Eds. AAAI Press, July 2015, pp. 1544–1550.

[10] M. Asai and A. S. Fukunaga, "Solving large-scale planning problems by decomposition and macro generation," in *Proceedings of the International Conference of Automated Planning and Scheduling (ICAPS)*, 2015.

[11] R. Finke and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 3-4, no. 2, pp. 189–208, 1971.

[12] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng, "VMSP: Efficient vertical mining of maximal sequential patterns," in *Canadian Conference on Artificial Intelligence*. Springer, 2014, pp. 83–94.

[13] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, "SPMF: A Java Open-Source Pattern Mining Library," *Journal of Machine Learning Research*, vol. 15, pp. 3569–3573, 2014.

[14] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *JAIR*, vol. 14, pp. 253–302, 2001.